Algorithmique et Structures de Données I

Dr. Maher Helaoui

2019 - 2020

Bibliographie

- Kernighan, B. et Ritchie, D.:
  The C Programming Language. (1988)
- Faber, F.:
  Cours : Programmation C.
  http://www.ltam.lu/cours-c/prg-c_c.htm, (1997)
- Ben Ayed, D.:
  Cours : Algorithmique et structures de données.
  ISI, (2018)
- Helaoui, M.:
  Travaux Dirigés : Algorithmique et Structures de Données.
  https://www.researchgate.net/publication/266392442, (2016)

Plan

Introduction à l'algorithmique et structures de données

Algorithmique : résoudre un problème par le calcul

Résoudre un problème par le calcul

Pendant plusieurs millénaires, les mathématiciens se sont contentés d'une notion intuitive, informelle, d'algorithme :

- une « méthode effective de calcul »,
- un « processus de résolution d'un problème par le calcul »

Les Baboliens



Figure: Le premier algorithme

Historique: les algorithmes

2000 ans avant J.C.

les Babyloniens (l'actuel Irak) ont marqué les premiers algorithmes sur terre et étaient principalement des méthodes de calcul pour le commerce et les impôts.

300 ans avant J.C.

les Grecs ont présenté l'algorithme D'Euclide pour le calcul du pgcd.

900 ans après J.C.

Al Khuwarizmi, un mathématicien perse (actuel Iran), consacre un ouvrage aux algorithmes.

Historique: les machines et l'automatisation du calcul

Entre 1600 et 1800

Pascal et Leibniz construisent des machines à calculer mécaniques (la Pascaline en 1642) ou des automates, et d'où l'automatisation du calcul et la recherche d'algorithmes efficaces.

La pascaline

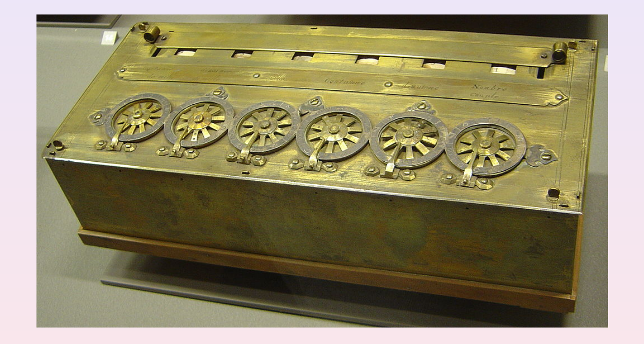


Figure: La Pascaline : calculatrice mécanique

Premier ordinateur

En 1928

Hilbert avec son Entscheidungsproblem (« problème de décision ») demande s'il existe un algorithme capable de décider si un énoncé mathématique est vrai ?
Une machine peut elle prendre une décision ?

En 1930 – 1937

Présentation de la machine de Turing.

En 1945

la construction des premiers ordinateurs inspirés de la machine de Turing.

Machine de Turing

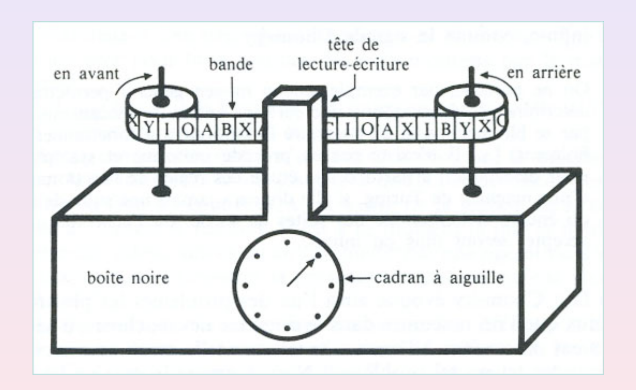


Figure: Machine de Turing

Section II. Définir la notion d'algorithmique

Algorithme

Définition

Un algorithme est une méthode générale pour résoudre un type ou une classe de problèmes. Il est dit correct lorsque qu'il résout le problème posé.

Efficacité

l'efficacité d'un algorithme est mesurée par sa durée de calcul, par sa consommation de mémoire vive, par la précision des résultats obtenus, etc.

Algorithmique

Définition

L'algorithmique est l'étude et la production de règles et techniques qui sont impliquées dans la définition et la conception d'algorithmes, c'est-à-dire de processus systématiques de résolution d'un problème permettant de décrire précisément des étapes pour résoudre un problème algorithmique.

L'algorithmique est la logique d'écrire des algorithmes.

Etapes de résolution d'un problème P

Résoudre P

- Définir
- Analyser
- Algorithme informel
- Conception
- Algorithme formel
- Implémentation et tests
- Algorithme codés
- Maintenance

Section III. Structure générale d'un algorithme

Structure générale d'un algorithme

Syntaxe de la structure générale

Algorithme <Nom de l'Algorithme>
<**Déclaration** de **constantes**>
<**Déclaration** de **types**>
<**Déclaration** de **variables**>
Début
<**Instructions**>
Fin

Il est possible d'ajouter des commentaires (des explications et des informations qui ne vont pas être exécutées ou prises en considération par l'ordinateur) :
/* commentaires */ (plusieurs lignes) ou
// commentaires (une seule ligne)

Section IV. Premier programme : instructions d'entrée/sortie

Instruction de sortie : d'affichage ou d'écriture

L'instruction d'écriture "Ecrire" affiche des informations sous une forme compréhensible sur un périphérique de sortie (L'écran). Ce qui permet au développeur de **communiquer** avec l'utilisateur de son programme.

Syntaxe de l'instruction de sortie

- Ecrire(Ident) ⇒ affiche sur l'écran le contenu de la variable Ident
- Ecrire(100) ⇒ affiche sur l'écran 100
- Ecrire("Bonjour tous les étudiants présents") ⇒ affiche sur l'écran le texte Bonjour tous les étudiants présents.
- Ecrire("Bonjour", X) ⇒ affiche sur l'écran le texte Bonjour puis le contenu de la variable X

Instruction d'entrée : de lecture

L'instruction de lecture "Lire" permet à l'utilisateur d'entrer des valeurs au programme à partir de l'entrée standard. (Le clavier).

Syntaxe de l'instruction de sortie

- Lire(Ident1, . . . , Identi, . . . , identn) $\Rightarrow$ saisir une valeur à partir du clavier et la mettre dans la variable Identi

Premier algorithme

Exercice

Ecrire un algorithme qui lit :

- le prix hors taxe d'un article
- le nombre d'articles achetés
- le taux de la TVA

Et qui affiche le montant total toute taxe comprise.

Premier algorithme

Solution

Algorithme montant

/* Cette partie de déclaration sera expliquée lors du prochain chapitre*/

PHT, TVA: Réel

Nbre: Entier

Début

Ecrire("Veillez donner le prix hors taxe d'un article")

Lire(PHT)

Ecrire("Veillez donner le nombre d'articles achetés")

Lire (Nbre) Ecrire("Veillez donner le taux de la TVA")

Lire (TVA)

Ecrire ("le montant total est:", (PHT*Nbre) * (1+(TVA*0.01)))

Fin

Merci pour votre attention
https://sites.google.com/site/maherhelaoui/

Chapitre II. Les variables

Section I. Introduction

Objectif ?

Maîtriser les notions : variable, type et valeur.

à la fin de ce chapitre vous devez

- faire la différence entre **variable** et **valeur**
- savoir **affecter** une valeur à une variable
- savoir accorder le bon **type** à une variable



Section II. Déclaration de variable

Besoin d'un programme

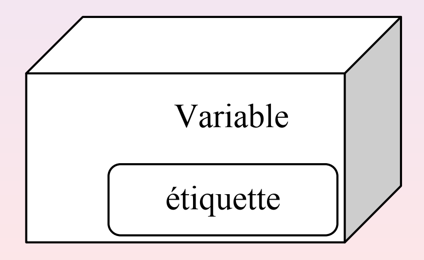
Mémoriser provisoirement des informations

Un programme a besoin de mémoriser provisoirement

- des **valeurs** (information)
- ces valeurs peuvent être de plusieurs **types** (des nombres, du texte, etc.)
- pour ce faire, nous pouvons utiliser des **variables**.

Schématiser une variable

Une variable peut être **schématisée** par une **boîte**, repérée par une **étiquette.** Pour avoir accès au contenu de la boîte, il suffit de la désigner par son étiquette.



Déclaration des variables

Une variable : un emplacement de mémoire, désigné par une adresse binaire

- Dans la **mémoire vive de l'ordinateur**, la boîte et l'étiquette sont remplacées par un **emplacement de mémoire**, désigné par une **adresse binaire**.
- Le **compilateur** se charge d'affecter une étiquette choisie par le programmeur pour chaque adresse binaire.

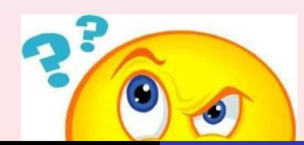
- Afin d'utiliser une variable il faut préparer son emplacement mémoire : créer la boîte et lui coller une étiquette.
- Ceci peut se faire au début de l'algorithme (**Variables globales**), avant même les instructions proprement dites. C'est ce qu'on appelle la **déclaration des variables**.

Type ?

C'est quoi un compilateur ?

Un programme écrit par le programmeur (en Langage structuré) ne peut être exécuté par votre ordinateur que s'il est transformé en langage compris par la machine : **Le langage binaire**.

Le compilateur est un **programme** permettant de **transformer un code écrit en Langage structuré en un code écrit en langage binaire.**



Syntaxe de la déclaration des variables

- **Variable** g : Entier Long
- **Variables** PrixHT, TauxTVA, PrixTTC : Réel Simple

Une variable est déclarée par la syntaxe suivante :
Variable Nom de la variable (étiquette ou identificateur) : Type de la variable

Type ?

C'est quoi le type d'une variable ?

Une bonne question, le type d'une variable est l'objet de la prochaine Section.



Section III. Type d’une variable et opérateurs associés

Type ?

C'est quoi le type d'une variable ?

- Pour créer une boîte (réserver un emplacement mémoire pour notre variable) nous devons préciser **sa taille**.
- Elle doit correspondre à ce que l'on voudra mettre dedans, il faut **éviter** :
  1. un **gaspillage** de mémoire de créer une boite (réserver un emplacement mémoire) plus grande par rapport à notre besoin.
  2. une **taille insuffisante** pour mémoriser toute l'information utile. (Perte de données utiles)

Optimiser la taille

Imaginons que nous allons construire un château pour élever un oiseau ou construire une toute petite cage pour un éléphant.

Types numériques

Types numériques classiques

- **Byte** (octet) de 0 à 255
- **Entier simple** de -32 768 à 32 767
- **Entier long** de -2 147 483 648 à 2 147 483 647
- **Réel simple** de -3,40E38 à -1,40E-45 pour les valeurs négatives et de 1,40E-45 à 3,40E38 pour les valeurs positives
- **Réel double** de -1,79E308 à -4,94E-324 pour les valeurs négatives et de 4,94E-324 à 1,79E308 pour les valeurs positives.

Opérateurs : Types numériques

Opérateurs : Types numériques classiques

- **Entier** : Parenthèse, Exposant, $\{+, -, *, /, \%, div\}$, $\{=, \neq, <, <=, >, >=\}$
- **Réel** : Parenthèse, Exposant, $\{+, -, *, /\}$, $\{=, \neq, <, <=, >, >=\}$

Types non numériques

exemples de types non numériques

- **alphanumérique** (également appelé **caractère**) lettres, de signes de ponctuation, d'espaces, ou de chiffres.
- **chaîne de caractères** (une telle chaîne de caractères est toujours notée entre guillemets) selon le type 2010 peut représenter le nombre 2010, ou la suite de caractères 2, 0, 1 et 0.
- **booléen** uniquement les valeurs logiques VRAI et FAUX

Opérateurs : Types non numériques

Opérateurs et fonctions

- **caractère** : $\{=, \neq, <, <=, >, >=\}$
  La comparaison entre les caractères se fait selon leur code ASCII ' ' $<$ '0' $<$ '7' $<$ 'A' $<$ 'Z' $<$ 'a' $<$ 'z' $<$ '$\}$'
  $\exists$ des fonctions prédéfinies comme Succ(Ident), Pred(Ident), ASCII(Ident)
- **booléen** : NON, ET, OU, $\{=, \neq, <, <=, >, >=\}$
- **Chaîne de caractères** : $\exists$ des procédures et des fonctions prédéfinies comme la fonctions LONG("Bonjour") renvoie la cardinalité de la chaîne dans cet exemple 7

Types et langage C

Type	Explanation	Format specifier
char	Smallest addressable unit of the machine that can contain basic character set. It is an integer type. Actual type can be either signed or unsigned. It contains CHAR_BIT bits.[3]	%c
signed char	Of the same size as char, but guaranteed to be signed. Capable of containing **at least** the [−127, +127] range;[3][note 1]	%c (or %hhi for numerical output)
unsigned char	Of the same size as char, but guaranteed to be unsigned. Contains **at least** the [0, 255] range.[4]	%c (or %hhu for numerical output)
short short int signed short signed short int	*Short* signed integer type. Capable of containing **at least** the [−32,767, +32,767] range;[3][note 1] thus, it is at least 16 bits in size. The negative value is −32767 (not −32768) due to the one's-complement and sign-magnitude representations allowed by the standard, though the two's-complementrepresentation is much more common.[5]	%hi
unsigned short unsigned short int	*Short* unsigned integer type. Contains **at least** the [0, 65,535] range;[3]	%hu
int signed signed int	Basic signed integer type. Capable of containing **at least**the [−32,767, +32,767] range;[3][note 1] thus, it is at least 16 bits in size.	%i or %d
unsigned unsigned int	Basic unsigned integer type. Contains **at least** the [0, 65,535] range;[3]	%u
long long int signed long signed long int	*Long* signed integer type. Capable of containing **at least** the [−2,147,483,647, +2,147,483,647] range;[3][note 1] thus, it is at least 32 bits in size.	%li
unsigned long unsigned long int	*Long* unsigned integer type. Capable of containing **at least**the [0, 4,294,967,295] range;[3]	%lu
long long long long int signed long long signed long long int	*Long long* signed integer type. Capable of containing **at least** the [−9,223,372,036,854,775,807, +9,223,372,036,854,775,807] range;[3][note 1] thus, it is at least 64 bits in size. Specified since the C99 version of the standard.	%lli
unsigned long long unsigned long long int	*Long long* unsigned integer type. Contains **at least** the [0, +18,446,744,073,709,551,615] range;[3] Specified since the C99 version of the standard.	%llu
float	Real floating-point type, usually referred to as a single-precision floating-point type. Actual properties unspecified (except minimum limits), however on most systems this is the IEEE 754 single-precision binary floating-point format(32 bits). This format is required by the optional Annex F "IEC 60559 floating-point arithmetic".	Converting from text:[a] %f %F %g %G %e %E %a %A
double	Real floating-point type, usually referred to as a double-precision floating-point type. Actual properties unspecified (except minimum limits), however on most systems this is the IEEE 754 double-precision binary floating-point format(64 bits). This format is required by the optional Annex F "IEC 60559 floating-point arithmetic".	%lf %lF %lg %lG %le %lE %la %lA[b]
long double	Real floating-point type, usually mapped to an extended precision floating-point number format. Actual properties unspecified. It can be either x86 extended-precision floating-point format (80 bits, but typically 96 bits or 128 bits in memory with padding bytes), the non-IEEE "double-double" (128 bits), IEEE 754 quadruple-precision floating-point	%Lf %LF %Lg %LG %Le %LE %La %LA[b]

Opérateurs et priorités

Ordre de priorité entre les opérateurs

Parenthèse $\succ$ Exposant $\succ$ Non $\succ \{*, /, \%, \mathit{div}\} \succ \{+, -\} \succ \{<, <=, >, >=\} \succ \{=, \neq\} \succ$ ET $\succ$ OU

NB : en cas d'égalité dans l'ordre de priorité la priorité est de gauche à droite

Tables de vérité (1)

Table de vérité de négation

Ident1	NON Ident1
vrai	faux
faux	vrai

Table de vérité : conjonction ET

Ident1	Ident2	Ident1 ET Ident2
vrai	vrai	vrai
vrai	faux	faux
faux	vrai	faux
faux	faux	faux

Tables de vérité (2)

Table de vérité : disjonction OU

Ident1	Ident2	Ident1 OU Ident2
vrai	vrai	vrai
vrai	faux	vrai
faux	vrai	vrai
faux	faux	faux

Section III. Instruction d'affectation

Syntaxe de l'instruction affectation

Syntaxe

Une variable permet de mémoriser une information, cette opération se fait à travers une instruction : **l'affectation** (lui attribuer une valeur).
En algorithmique, cette instruction se note avec le signe $\leftarrow$.

Syntaxe de l'instruction affectation :
Nom de la variable $\leftarrow$ Valeur de la variable.

Exemple déclaration

Exemple

Variable Boite : Entier

/*Cette ligne permet de réserver un espace mémoire suffisant pour contenir un entier au niveau de la déclaration notre boite est vide.*/

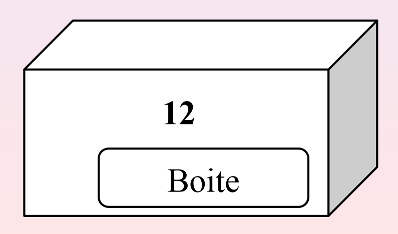


Exemple affectation

Exemple

Boite ← 12

/*Cette ligne permet d'affecter la valeur entière de 12 à notre variable Boite.*/



Exemple réaffectation

Exemple

Boite ← 20

/*Cette ligne permet d'écraser la valeur de 12 dans notre variable Boite et affecter une nouvelle valeur entière de 20.*/



Des questions ?

Questions :

1. Et les **constantes** ?
2. Pourquoi avez-vous confondu **Bit**[a] et **octet** ?
3. Est-il possible d'**allouer x/8 octet**[b] avec un langage structuré comme le C ?
4. Pourquoi le type **booléen** n'est pas **numérique** ? Il n'existe pas en C ?
5. Quel est l'**espace mémoire** réservé au **type booléen** ?

[a] The byte is a unit of digital information that most commonly consists of eight bits ... The size has historically been hardware dependent

[b] Le Byte est la plus petite unité ≪ logiquement ≫ adressable par un programme

Les constantes

Une constante

Une **constante** est **une variable dont la valeur est fixée tout au long du programme.**

Syntaxe constante

Constante
Nom de la constante ← Valeur de la constante : Type

L'enregistrement

Créer mon propre type

L'**enregistrement** est le fait de **créer une nouvelle structure de données en créant un nouveau type**.

Syntaxe

Type
Nom de la structure : Enregistrement
Structures de données Typés
Fin Enregistrement

Créer un enregistrement

Booléen

1. Supposons que nous n'avons pas de type booléen, ce qui est le cas avec le langage c, et nous voulons le créer.
2. quelle est l'allocation mémoire à réserver pour le nouveau type créé ?

Exercice

Exercice 1

Soit l'algorithme Test1 suivant :

Algorithme Test1

// Ajouter les déclarations nécessaires

Début

A $\leftarrow$ 4(1)

B $\leftarrow$ 11(2)

A $\leftarrow$ $B - A(3)$

B $\leftarrow$ $B - A(4)$

A $\leftarrow$ $A + B(5)$

Fin

- Ajouter les déclarations nécessaires à l'algorithme Test1
- Que fait l'algorithme ci-dessus ?

Exercice 1 suite

Exercice 1 suite

- Ce résultat est-t-il toujours vrai ? Etablir la trace de cet algorithme (sous forme de tableau) avec a et b pour valeurs initiales de A et B.
- Ecrire un nouvel algorithme équivalant en n'utilisant que des variables ? (Utiliser une variable intermédiaire).

Section IV. Conclusion

Ce qu'il faut retenir

Variable, type et valeur.

à la fin de ce chapitre vous devez

- faire la différence entre **variable** et **valeur**
- savoir **déclarer** une variable
- savoir utiliser l'**instruction d'affectation**
- Gérer la déclaration des structures de données.

Merci pour votre attention
https://sites.google.com/site/maherhelaoui/

Chapitre III. Les structures de contrôle

Section I. Introduction

Pourquoi ?

Objectifs ?

Ce chapitre a pour objectifs de

- Définir et présenter la syntaxe des structures de contrôle
- Maîtriser et savoir utiliser les structures de contrôle

Section II. Les Instructions conditionnelles

Formes de l'instruction **conditionnelle Si**

Syntaxes des divers formes

1. **Si** (expression logique) **Alors** Instructions **FinSi**
2. **Si** (expression logique 1) **Alors** Instructions 1 **Sinon** Instructions 2 **FinSi**
3. // Les conditionnelles emboîtées :
   Si (expression logique 1) **Alors** Instructions 1
   Sinon Si (expression logique 2) **Alors** Instructions 2
   . . .
   Sinon Si (expression logique n-1) **Alors** Instructions n-1
   Sinon Instructions n
   FinSi

Exemple : instruction conditionnelle Si

Instruction conditionnelle Si

Ecrire un programme permettant à un étudiant de l'ISI de saisir sa moyenne et il affiche le résultat et la mention obtenue.

Syntaxe de l'instruction **Selon**

Nous pouvons remplacer Les conditionnelles emboîtées par l'instruction **Selon** (permet une facilité d'écriture)

Syntaxe

Selon identificateur **Faire**
(liste de) valeur(s) 1 : instructions 1
(liste de) valeur(s) 2 : instructions 2
...
(liste de) valeur(s) n-1: instructions n-1
Sinon
instructions n
FinSelon

Exemple : instruction conditionnelle Selon

Instruction conditionnelle Selon

Ecrire un programme permettant à un étudiant de l'ISI de saisir sa moyenne et selon la moyenne il affiche le résultat et la mention obtenue.

Section III. Les structures itératives

La structure itérative **Pour**

Syntaxe

1. **Pour** compteur de début à fin [(pas = val)] **Faire**
   Instructions
   FinPour

Exemple : structure itérative Pour

structure itérative Pour

Ecrire un programme permettant à un étudiant de l'ISI de proposer cinq idées permettant à son avis d'améliorer la qualité des formations.

La structure itérative **Tant que**

Syntaxe

1. **Tantque** (expression logique) **Faire**
   Instructions
   FinTantque

Exemple : structure itérative Tant que

structure itérative Tant que

Ecrire un programme permettant à un étudiant de l'ISI de proposer toutes idées permettant à son avis d'améliorer la qualité des formations.

La structure itérative **Répéter Jusqu'à**

Syntaxe

1. **Répéter**
   Instructions
   Jusqu'à (expression logique)

Exemple : structure itérative Répéter Jusqu'à

structure itérative Répéter Jusqu'à

Ecrire un programme permettant à un étudiant de l'ISI de proposer au minimum une idée permettant à son avis d'améliorer la qualité des formations.

Section V. Conclusion

Conclusion

Conclusion

Dans ce Chapitre nous avons présenté:

- Les **instructions conditionnelles.**
- Les **structures itératives**

Merci pour votre attention
https://sites.google.com/site/maherhelaoui/

Chapitre IV. Les Sous programmes et Tableaux

Section I. Introduction

Pourquoi ?

Objectifs ?

Ce chapitre a pour objectifs de

- Maîtriser la programmation modulaire
- Introduire les tableaux et les matrices
- Présenter des algorithmes de tri

Section II. Les Sous programmes

Exemple 1

Pourquoi les sous programmes ?

Pouvez vous écrire un programme qui

- demande à l'utilisateur de remplir les valeurs de deux variables de types entiers $X1$ et $Y2$?
- calcule la somme de $X1$ et $Y2$?
- demande à l'utilisateur de remplir deux autres variables de types entiers $X2$ et $Y3$?
- calcule la somme de $X2$ et $Y3$?
- demande à l'utilisateur de remplir deux autres variables de types entiers $X3$ et $Y4$?
- calcule la moyenne de $X3$ et $Y4$?

Exemple 2

Pourquoi les sous programmes ?

Pouvez vous écrire un programme qui

- demande à l'utilisateur de remplir deux variables $X1$ et $Y2$?
- permute les valeurs de $X1$ et $Y2$?
- demande à l'utilisateur de remplir deux autres variables $X2$ et $Y3$?
- permute les valeurs de $X2$ et $Y3$?
- demande à l'utilisateur de remplir deux autres variables $X3$ et $Y4$?
- permute les valeurs de $X3$ et $Y4$?

Procédure et fonction

Procédure

Une procédure est un sous programme qui n'a pas de retour propre à la procédure.

Fonction

Une fonction est une procédure qui a un type et un retour, de même type que la fonction, une ou un ensemble de valeurs, propres à la fonction.

Syntaxe d'une procédure

Syntaxe

Nom de la procédure(liste typée de paramètres formels)
/*en-tête de la procédure*/
Déclaration des variables locales
Début
Instructions /*Un bloc homogène*/
Fin

Syntaxe d'une fonction

Syntaxe

Nom de la fonction(liste typée paramètres formels) : [**type propre à la fonction**]/*en-tête de la fonction*/
Déclaration des variables locales
Début
Instructions /*Un bloc homogène*/
Nom de la fonction $\leftarrow$ expression ou variable à renvoyer /*Mot Clé du retour*/
Fin

Structure générale d'un algorithme

Mise à jour de la Syntaxe de la structure générale

Algorithme <Nom de l'Algorithme>
<**Déclaration** de **constantes**>
<**Déclaration** de **types**>
<**Déclaration** de **variables**>
<**Procédures et Fonctions**>
Début
<**Instructions**>
Fin

Instruction d'appel d'une fonction ou une procédure
Nom de la procédure(Liste de paramètres non typée)
Variable ← Nom de la fonction(Liste de paramètres non typée)

Fonctions somme et moyenne

Fonction somme de deux entiers

Somme(A:Entier, B:Entier) : Entier/*en-tête de la fonction somme*/
Début
Somme ← A+B
Fin

Fonction moyenne de deux entiers

Moyenne(A:Entier, B:Entier) : Réel /*en-tête de la fonction Moyenne*/
Somme(A:Entier, B:Entier) : Entier /*La déclaration de la fonction Somme n'est pas obligatoire*/
Début
Moyenne ← Somme(A,B)/2

Exercice 1

Solution Exercice 1

Algorithme Solution-Exercice-1
Somme(A:Entier, B:Entier) : Entier/*en-tête de la fonction somme*/
Début Somme ← A+B **Fin**
Moyenne(A:Entier, B:Entier) : Réel /*en-tête de la fonction Moyenne*/
Début Moyenne ← Somme(A,B)/2 **Fin**
i, a, b : Entiers c: Réel
Début
Pour i de 1 à 3 **Faire**
Ecrire("Donnez un entier") Lire(a)
Ecrire("Donnez un entier") Lire(b)
Si (i $<$ 3) **alors** Ecrire("La somme est : ", Somme(a,b)) **FinSi**

Passage de paramètres par valeurs et par adresses

Passage de paramètres par valeurs

Nous passons le contenu de la variable en paramètre : le contenu de la variable passée en paramètre ne sera pas modifié suite à l'appel de la fonction ou la procédure.

Passage de paramètres par adresses

Nous passons l'adresse physique de la variable en paramètre : le contenu de la variable passée en paramètre sera modifié suite à l'appel de la fonction ou la procédure.

Passage de paramètre par adresse

Var Boite xxxxxx



L'Adresse de la variable est son emplacement mémoire physique.

Exercice 2

Solution Exercice 2

Algorithme Solution-Exercice-2
Permuter(Var A:Entier, Var B:Entier) /*en-tête de la Procédure Permuter*/
temps : Entier **Début**
temps ← A
A ← B
B ← temps
Fin
a, b,c,d,e,f : Entiers
Début
Ecrire(”Donnez un entier X1”) Lire(a)
Ecrire(”Donnez un entier Y1”) Lire(b)
Ecrire(”Donnez un entier X2”) Lire(c)

Appel d'une fonction (1)

Fonctions ne renvoyant rien au programme et sans passage d'arguments

Une fonction ne renvoyant rien au programme est une Procédure de type void.

Fonction renvoyant une valeur au programme et sans passage d'arguments

Une fonction ne possède pas de paramètre formel et après exécution, renvoie une valeur. Le type de cette valeur est déclaré avec la fonction. La valeur retournée est spécifiée à l'aide du mot réservé return.

Fonction avec passage d'arguments

Fonctions utilisent les valeurs de certaines variables du

Appel d'une fonction (2)

Fonction avec passage d'arguments

Fonctions utilisent les valeurs de certaines variables du programme les ayant appelé.

Exemple 4

Fonction ne renvoyant rien au programme et sans passage d'arguments

Appelons une fonction carre ne renvoyant rien au programme et sans passage d'arguments pemettant de calculer le carré d'un entier choisi par l'utilisateur de notre programme.

Exemple 4 bis

Fonction renvoyant une valeur au programme et sans passage d'arguments

Appelons une fonction carre renvoyant au programme le carré d'un entier choisi par l'utilisateur et sans passage d'arguments.

Exemple 4 bis bis

Fonction avec passage d'arguments

Appelons une fonction carre renvoyant au programme le carré d'un entier choisi par l'utilisateur au programme principale et passons le comme argument à cette fonction.

Nous pouvons appeler la fonction carre autant de fois que l'on veut avec des variables différentes.
x est un paramètre formel ou argument : ce n'est pas une variable du programme.
$n1$ une variable du programme est un paramètre effectif de la fonction carre.

Le passage de paramètres

Le passage de paramètres d'une fonction

Avec l'instruction return d'une part et la liste des paramètres d'autre part, une fonction dispose de deux moyens pour communiquer ou pour échanger des données avec d'autres fonctions.

Divers passage de paramètres d'une fonction

- Passage de paramètres par valeur.
- Passage de paramètres par adresse.

Exemple : Passage de paramètres par valeur d'une fonction

Passage de paramètres par valeur

Reprenons l'Exemple 3 et présentons

- une première solution avec passage de paramètres par valeur.
- une deuxième solution avec passage de paramètres par adresse.

Section III. Les Tableaux et les Matrices

Tableaux ?

C'est quoi un tableau ?

Un tableau ou vecteur T est une suite de N variables ou enregistrement de même types (appelées éléments) dont les adresses sont ordonnées.
N est la dimension de T : elle nous permet de connaître l'espace mémoire consommé par T.

Adresses ordonnées ?

Pourquoi les adresses sont ordonnées ?

L'ordre sur les adresses de T permet l'accès à n'importe quel élément $T[i]$ de T étant donnée son indice et l'adresse du premier élément.

Indice ?

Les indices est un ordre sur les adresses des divers éléments de T.

1. Le premier élément est d'indice 1 : accessible par T[1]
2. Le deuxième élément est d'indice 2 : accessible par T[2]
3. ...
4. Le nième élément est d'indice n : accessible par T[n]

Syntaxe des Tableaux

Déclaration

Nom Tableau : Tableau [premier indice . . . dernier indice]de Type élément

Tableaux à 2 dimentions ?

C'est quoi une Matrice ?

Un tableau à deux dimensions ou Matrice M est une suite de L Tableaux chacun de dimension C de même types dont les adresses sont ordonnées.
L'espace mémoire consommé par M est de $L * C$ de type d'un élément.

Adresses ordonnées ?

Pourquoi les adresses sont ordonnées ?

L'ordre sur les adresses de M permet l'accès à n'importe quel élément $M[i][j]$ de M étant donnée ses indices et l'adresse du premier élément.

Indice ?

Les indices est un ordre sur les adresses des divers éléments de M.

1. Le premier élément est d'indice 1ere Ligne 1ere Colonne : accessible par M[1][1]
2. Le deuxième élément est d'indice 1ere Ligne 2eme Colonne : accessible par M[1][2]
3. ...

Syntaxe des Tableaux

Déclaration

Nom Matrice : Tableau [1er indice . . . *L*eme indice] [1er indice . . . *C*eme indice] de Type élément

Section IV. Les Algorithmes de Tri

Tri par sélection

Principe

1. Permuter l'élément le plus petit de T[1 . . . N] avec l'élément 1 de *T*
2. Permuter l'élément le plus petit de T[2 . . . N] avec l'élément 2 de *T* . . .
3. Permuter l'élément le plus petit de T[N-1, N] avec l'élément N-1 de *T*

Pour i de 1 à N-1 **Faire**
Permuter l'élément le plus petit de T[i . . . N] avec l'élément i de *T*
FinPour

Tri par sélection

Algorithme : Tri par sélection
Const
NMax $\leftarrow$ 1000 : *Entier*
Variables
i,N, IndicePP : Entiers
T: Tableau [1 ... NMax] d'Entiers
Debut
Pour i de 1 à N-1 **Faire**
IndicePP $\leftarrow$ $SelectionnerLePlusPetit(T[], i, N)$
Permuter(T,i,IndicePP)
FinPour
Fin

Sélectionner Le Plus Petit

SelectionnerLePlusPetit(T[]:Entier,i:Entier,N:Entier) : Entier
Variables
j,IndiceMin : Entiers
Debut
IndiceMin $\leftarrow i$
Pour j de i+1 à N **Faire**
Si T[j] $<$ T[IndiceMin] **Alors**
IndiceMin $\leftarrow j$
FinSi
FinPour
SelectionnerLePlusPetit $\leftarrow IndiceMin$
Fin

Permuter Le Plus Petit

Permuter(T:Entier,i,IndicePP)/*T:Entier est équivalente à Var T[]*/
Début
Si i $\neq IndicePP$ **Alors**
Variable
Aux : Entier
Aux $\leftarrow T[i]$
T[i] $\leftarrow T[IndicePP]$
T[IndicePP] $\leftarrow Aux$
FinSi
Fin

Tri à bulle

Principe

Tantque une permutation est possible **Faire**
Permuter les plus grands éléments du tableau enfin de tableau.
FinTantque

Ceci est équivalent à
Tantque une permutation est possible **Faire**
Permuter les plus petits éléments du tableau au début du tableau.
FinTantque

Tri à bulle

Procédure TRISBULLE (VAR T : tableau entier, N : entier)
Variables i, j, aux : Entiers
DEBUT
Pour i de 1 à N-1 **Faire**
j $\leftarrow$ N
Tantque (j $\neq$ i) **Faire**
Si (T[j] $<$ T[j - 1]) **alors**
Permuter(T,j,j-1)
Finsi
j $\leftarrow$ j 1
FinTantque
FinPour
FIN

Tri par insertion

Procédure TriInsertion(T:tableau Entier, n: Entier)
Variables
x,i,j:Entiers
pour (i de 1 à n - 1) **Faire**
x $\leftarrow$ T[i]
j $\leftarrow$ i
Tant que (j $>$ 0 et T[j - 1] $>$ x) **Faire**
T[j] $\leftarrow$ T[j - 1]
j $\leftarrow$ j - 1
FinTantque
T[j] $\leftarrow$ x
FinPour
Fin

Section V. Conclusion

Conclusion

Conclusion

Dans ce Chapitre nous avons présenté:

- Les Sous programmes.
- Les Tableaux et les Matrices.
- Les Algorithmes de Tri.

Merci pour votre attention

https://sites.google.com/site/maherhelaoui/

PUBLICATIONS

Bibliographie

Kernighan, B. et Ritchie, D.:
The C Programming Language. (1988)

Faber, F.:
Cours : Programmation C.
http://www.ltam.lu/cours-c/prg-c_c.htm, (1997)

Ben Ayed, D.:
Cours : Algorithmique et structures de données.
ISI, (2018)

Helaoui, M.:
Travaux Dirigés : Algorithmique et Structures de Données.

MERCI POUR VOTRE ATTENTION